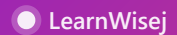


Ejercicio de programación con IA

Construye el ejemplo OpsMonitor de este módulo con ChatGPT o Claude a partir de un prompt listo para usar que dirige al modelo a la especificación del módulo, y después revisa, ejecuta y amplía lo que te devuelva — incluso comparándolo con nuestra propia solución de referencia en GitHub.



Escribir el código ya es la mitad fácil.

Conseguir que un asistente moderno genere código que funcione para «Memoria, asignaciones y fugas de sesión» solo requiere un buen prompt. Juzgarlo, corregirlo y ampliarlo requiere todo lo que ha cubierto este módulo. Ese es el reparto sobre el que se construye este ejercicio: tú aportas la especificación y la revisión; el asistente, el tecleo.

Qué estás pidiendo

Ataca las dos mitades del problema de memoria en WisejPerfLab. Ejecuta el escenario de búsqueda de tickets con .NET Object Allocation, encuentra los modelos de fila y las cadenas asignadas en cada búsqueda y redúcelos proyectando una sola vez en lugar de reconstruir objetos en cada redibujado. Después toma un snapshot de Memory Usage tras el calentamiento, abre y cierra el formulario de detalle del ticket cincuenta veces, toma un segundo snapshot y compáralos: los formularios retenidos están enraizados por una suscripción estática a GlobalTicketBus.TicketChanged y por un temporizador de actualización que nunca se detuvo. Cancela la suscripción y detén ambos en un Dispose sobrescrito, limpia el origen de datos del BindingSource y vuelve a ejecutar el escenario de apertura/cierre para mostrar que el heap retenido vuelve a acercarse al snapshot A.

El módulo pide estos entregables, y el prompt de abajo le pide al asistente exactamente esos:

- ✓ Traza de asignaciones que nombra el tipo con más asignaciones y la ruta de llamadas del escenario de búsqueda, antes y después
- ✓ Comparación de los snapshots A/B que muestra el número de instancias retenidas del formulario de detalle antes de la corrección
- ✓ Captura de la ruta a la raíz que identifica la suscripción al evento estático y el temporizador activo
- ✓ Sobrescritura de Dispose que cancela la suscripción al evento estático, detiene y libera el temporizador y limpia el enlace de datos
- ✓ Presupuesto de memoria por sesión y lista de comprobación de liberación, con el snapshot «después» que demuestra que la ruta de retención ha desaparecido

Por qué tiene esta forma

El dashboard de WisejPerfLab ya cumple su presupuesto de CPU, y aun así la aplicación se viene abajo tras cuatro horas en producción. Los problemas de memoria se comportan de forma distinta a los de CPU: son

invisibles en una demo de dos minutos, nunca aparecen como un clic lento, y son la razón de que un servidor que aguanta sin esfuerzo ochenta sesiones por la mañana rechace conexiones a media tarde. En el laboratorio hay dos, y no son el mismo problema. La búsqueda de tickets asigna mucha más memoria de la que necesita, lo que cuesta tiempo de GC y CPU mientras se ejecuta. El formulario de detalle del ticket nunca se libera de verdad cuando el usuario lo cierra, lo que cuesta memoria para siempre. Uno se encuentra con una traza de asignaciones; el otro, solo con dos snapshots y una ruta a la raíz.

Estos son los términos que el asistente tiene que dominar. Si lo que te devuelve los usa a la ligera, no ha entendido el módulo:

Término	Qué significa aquí
presión de asignación	Cuánta memoria crea un escenario mientras se ejecuta; una presión alta aumenta la frecuencia del GC y roba CPU aunque no haya ninguna fuga.
memoria retenida	Lo que sigue siendo alcanzable cuando el escenario debería haber terminado; es lo que se multiplica por el número de sesiones y decide la capacidad.
.NET Object Allocation	La herramienta que registra cada asignación administrada durante un escenario y la agrupa por tipo y por la ruta de llamadas que asigna.
snapshots de Memory Usage	Capturas del heap tomadas en momentos elegidos; comparar el snapshot A y el snapshot B muestra qué tipos ganaron instancias que nunca se liberaron.
ruta a la raíz	La cadena de referencias que mantiene vivo un objeto; es la evidencia real de una fuga y la única forma de demostrar que ha desaparecido.
raíz de retención	Lo que está en lo alto de esa cadena: un campo estático, un evento estático, un temporizador en marcha, una tarea en segundo plano o una caché de larga vida.
presupuesto de memoria por sesión	Una cifra, en megabytes retenidos por sesión inactiva, que convierte «la aplicación usa mucha memoria» en un aprobado o un suspenso y, más adelante, en un umbral del health check.

Constrúyelo — elige tu asistente

Los dos botones abren una pestaña nueva con el **mismo prompt listo para usar** ya escrito. El prompt dirige al modelo a la especificación de este módulo — un único archivo Markdown con la lección, el enunciado del laboratorio y los entregables — para que trabaje con el material real en lugar de adivinar de qué trata el Módulo 4. Puedes abrir tú mismo [module-04.md](#) para ver exactamente lo que leerá.



Usar ChatGPT

Abre [chatgpt.com](#) con el prompt de este módulo ya cargado. Funciona con GPT-5 o



Usar Claude

Abre [claude.ai](#) con el mismo prompt ya cargado. Si usas Claude Code, puedes

Codex; cualquier plan que pueda navegar por la web descargará la especificación.

pegarlo en una terminal abierta en una carpeta vacía.



Envíalo tal cual y obtendrás un proyecto, no un fragmento

El prompt termina pidiendo toda la solución como un único `.zip` con los pasos para compilarla y ejecutarla. Descárgalo, abre la solución en Visual Studio, pulsa F5 y OpsMonitor se abrirá en `localhost`. Necesitas el SDK de .NET y el paquete NuGet `Wisej-4`; todo lo demás, el prompt le pide al asistente que lo escriba él mismo.

Estoy siguiendo el Módulo 4 - «Memoria, asignaciones y fugas de sesión» - del curso de LearnWisej «Rendimiento y profiling», y quiero construir contigo el ejemplo de este módulo. Respóndeme en español.

Empieza leyendo la especificación del módulo. Es un único archivo Markdown (en inglés) que contiene la lección completa, el enunciado del laboratorio y los entregables:

<https://learnwisej.com/assets/courses/performance-and-profiling/specs/module-04.md>

Después construye un proyecto Wisej.NET 4 completo que pueda abrir en Visual Studio:

1. Proyecto «OpsMonitor», C#, net8.0 o posterior, paquete NuGet Wisej-4 4.1.x, host Kestrel que llame a `app.UseWisej()`.
2. Construye lo que pide el laboratorio de la especificación: Ataca las dos mitades del problema de memoria en `WisejPerfLab`. Ejecuta el escenario de búsqueda de tickets con `.NET Object Allocation`, encuentra los modelos de fila y las cadenas asignadas en cada búsqueda y redúcelos proyectando una sola vez en lugar de reconstruir objetos en cada redibujado. Después toma un snapshot de Memory Usage tras el calentamiento, abre y cierra el formulario de detalle del ticket cincuenta veces, toma un segundo snapshot y compáralos: los formularios retenidos están enraizados por una suscripción estática a `GlobalTicketBus.TicketChanged` y por un temporizador de actualización que nunca se detuvo. Cancela la suscripción y detén ambos en un `Dispose` sobrescrito, limpia el origen de datos del `BindingSource` y vuelve a ejecutar el escenario de apertura/cierre para mostrar que el heap retenido vuelve a acercarse al snapshot A.
3. Genera estos entregables e indícame dónde está cada uno dentro de la solución:
 - Traza de asignaciones que nombra el tipo con más asignaciones y la ruta de llamadas del escenario de búsqueda, antes y después
 - Comparación de los snapshots A/B que muestra el número de instancias retenidas del formulario de detalle antes de la corrección
 - Captura de la ruta a la raíz que identifica la suscripción al evento estático y el temporizador activo
 - Sobrescritura de `Dispose` que cancela la suscripción al evento estático, detiene y libera el temporizador y limpia el enlace de datos
 - Presupuesto de memoria por sesión y lista de comprobación de liberación, con el snapshot «después» que demuestra que la ruta de retención ha desaparecido
4. Sigue la especificación siempre que nombre una clase, propiedad o evento de Wisej.NET - usa exactamente ese. Donde deje la elección abierta, dime qué API elegiste y por qué.
5. Mantén la autoridad en el servidor: el estado y la validación viven en .NET, y el navegador muestra e informa. Nunca traslades una decisión a JavaScript porque allí resulte más fácil.
6. Haz que el resultado se pueda demostrar: debo poder ejecutarlo y ver el comportamiento del que trata este módulo, incluido lo que ocurre cuando algo sale mal.

Antes de escribir una sola línea de código, explica con tus propias palabras qué vas a construir y dónde vivirá cada pieza, para que pueda corregir el diseño mientras todavía es barato.

Cuando el código esté terminado, empaqueta toda la solución en un único archivo .zip descargable y dame los pasos exactos para compilarla y ejecutarla.

Por último, responde a estas preguntas de la especificación usando tu propia implementación como prueba:

- Tu traza de asignaciones mejoró muchísimo, pero la comparación de snapshots no ha

cambiado. ¿Qué tipo de problema corregiste, cuál no, y cómo notaría la diferencia un usuario?

– ¿Por qué el laboratorio abre y cierra el formulario cincuenta veces y no una, y qué concluirías si en el snapshot B apareciera una sola instancia?

– El callback del temporizador ocupa una línea y solo asigna una etiqueta. Explica con precisión cómo mantiene vivo un formulario entero, y qué resuelve y qué no una comprobación de `IsDisposed`.

Pregúntame por cualquier cosa ambigua antes de empezar.

Después revisa lo que te devuelve

Que el código compile no significa que sea correcto. Las preguntas interesantes son de diseño: ¿es este el control de Wisej.NET que la tarea realmente necesita?, ¿sigue el ciclo de vida que te ofrece el framework?, ¿el estado de referencia está en el servidor o se quedó en el navegador porque era más cómodo? Lee el resultado como revisor, no como cliente.

Este módulo te da la lista de comprobación. Responde a cada punto basándote en el código que acabas de recibir:

- ✓ Tu traza de asignaciones mejoró muchísimo, pero la comparación de snapshots no ha cambiado. ¿Qué tipo de problema corregiste, cuál no, y cómo notaría la diferencia un usuario?
- ✓ ¿Por qué el laboratorio abre y cierra el formulario cincuenta veces y no una, y qué concluirías si en el snapshot B apareciera una sola instancia?
- ✓ El callback del temporizador ocupa una línea y solo asigna una etiqueta. Explica con precisión cómo mantiene vivo un formulario entero, y qué resuelve y qué no una comprobación de `IsDisposed`.

Todo lo que hayas tenido que corregir merece quedar por escrito. Una nota breve con las convenciones y decisiones que acordaste, pegada en el primer mensaje de tu próxima sesión, vale más que cualquier prompt ingenioso.



Este ya lo construimos — compara tu versión con la nuestra

La implementación de referencia del Módulo 4 está en el repositorio público de ejemplos. Construye primero la tuya y después abre la nuestra: lo interesante son las diferencias.



[Abrir el ejemplo del Módulo 4 en GitHub](#)

O clona el conjunto completo con `git clone https://github.com/iceteagroup/LearnWisej-Samples` y abre `Performance and Profiling Course/Module 4`.



Nuestro ejemplo es un punto de partida, no un producto terminado

Se construyó igual que estás construyendo el tuyo — con un prompt y luego corregido — y se detiene a propósito donde se detiene el módulo. Espera soluciones provisionales donde iría código de producción, un manejo de errores mínimo y ninguna prueba. Tómallo como una invitación: usa lo que aprendiste en este módulo, junto con el asistente, para llevarlo más lejos. Mejorar código que no escribiste tú es precisamente la habilidad que enseña este curso.

Tres prompts para ir más allá

Dirige tu asistente al proyecto que acabas de construir — o al nuestro y prueba estos. Cada uno es un ejercicio de revisión y ampliación, no una reescritura.

- ✓ **Cambia una pieza sin mover las juntas.** *"Elige la parte de este proyecto OpsMonitor que sea más claramente un sustituto de algo real — una fuente de datos falsa, un almacén en memoria, un servicio simplificado — y reemplázala por una implementación con forma de producción. El contrato a ambos lados no debe cambiar: los mismos tipos, las mismas llamadas, el mismo comportamiento ante fallos. Dime qué te obligó a gestionar la implementación real que el sustituto no gestionaba."*
- ✓ **Haz que aguante un mal día.** *"Revisa este proyecto OpsMonitor en busca de los modos de fallo que todavía no gestiona — una entrada incorrecta o ausente, una dependencia lenta o no disponible, trabajo que llega después de que el usuario haya salido de la pantalla y dos acciones que compiten entre sí. Para cada uno, dime primero si es un riesgo real en Wisej.NET 4 y por qué, y solo después escribe la corrección. Si decides que algo no necesita gestión, dilo y explica el razonamiento en lugar de añadir código defensivo."*
- ✓ **Añade el segundo caso.** *"Amplía este proyecto OpsMonitor con un segundo caso que siga el mismo patrón que el primero — otra entidad, pantalla o control con la misma forma, conectado de la misma manera. Extrae a una clase base o a un servicio lo que de verdad sea común, pero solo donde se justifique; dos cosas parecidas no siempre piden una abstracción. Muéstrame el diff y justifica cada cosa que hayas extraído."*



Llévalo al laboratorio

El laboratorio práctico que cierra este módulo pide los mismos entregables que acabas de hacer que generara el asistente. Usa lo que tienes aquí como forma de tu respuesta y como prueba que la respalda — no como la respuesta en sí. El laboratorio se califica por tu razonamiento, y esa parte tiene que ser tuya.